

TOCALib: Optimal control library with interpolation for bimanual manipulation and obstacles avoidance

Yulia Danik¹, Dmitry Makarov², Aleksandra Arkhipova³, Sergei Davidenko⁴ and Aleksandr Panov⁵

Abstract—The paper presents a new approach for constructing a library of optimal trajectories for two robotic manipulators, Two-Arm Optimal Control and Avoidance Library (TOCALib)¹. The optimisation takes into account kinodynamic and other constraints within the FROST framework. The novelty of the method lies in the consideration of collisions using the DCOL method, which allows obtaining symbolic expressions for assessing the presence of collisions and using them in gradient-based optimization control methods. The proposed approach allowed the implementation of complex bimanual manipulations. In this paper we used Mobile Aloha as an example of TOCALib application. The approach can be extended to other bimanual robots, as well as to gait control of bipedal robots. It can also be used to construct training data for machine learning tasks for manipulation.

I. INTRODUCTION

The field of robotics is rapidly evolving, with a growing emphasis on the development and deployment of bimanual robots, such as the TIAGo [1] and anthropomorphic robots like Baxter [2], Figure 01, Atribot S1 etc. These robots, designed with two arms to mimic human coordination, have opened up new possibilities to perform complex tasks that require simultaneous control of multiple manipulators. Bimanual tasks are becoming increasingly important as industries seek more sophisticated automation solutions that can handle tasks requiring precise coordination and flexibility, such as assembly, packaging, and collaborative work with humans. Collision avoidance in robot manipulators is essential for safe and efficient operation. To accomplish this task in real time, it is necessary to plan the collision-free paths and control trajectories of the manipulators from the current positions to the desired positions that are comfortable for safe object grasping.

The recent emergence of relatively low-cost mobile bimanual robots has attracted considerable interest from researchers in this field. One of these robots is a Mobile Aloha, which was developed in 2024. It is a mobile platform with two independently controlled wheels equipped with four manipulators

(two rare for teleoperation and two front for manipulation). Its manipulators feature 16 DOF in total, with 8 DOF for each arm [3].

Usually Aloha is controlled using behavior cloning based on the Action Chunking with Transformers approach [3]. However, it is still relevant to control Mobile Aloha using alternative methods capable of solving tasks without human intervention and finding optimal or unconventional solutions. These include "classical" optimal control methods and reinforcement learning (RL) approaches. For the implementation of offline RL training or for the practical application of certain online RL methods (e.g. SERL [4]), high-quality task solution examples are required.

Thus, for both classical real-time optimal control manipulation and RL-based manipulation, it is desirable to have good initial approximations for the motion of the manipulators. To achieve this, a precomputed motion library has been proposed. The library contains optimal trajectories and controls for the manipulators, taking into account self-collisions and collisions with static and dynamic objects. Each trajectory is parameterised by the initial and final state of the manipulator.

This paper proposes a method for constructing a pre-computed Two-Arm Optimal Control and Avoidance Library (TOCALib) for the manipulators. We use the Mobile Aloha as an example of TOCALib application. The advantages of the library are following:

- Optimal trajectories and controls taking into account detailed kinodynamic model of the robot.
- Solution for arbitrary endpoints through the interpolation mechanism (trilinear interpolation) and the use of library entries.
- Reduction of computation and search time for trajectories between library grid nodes (no need to solve the optimization problem for new points).

The developed software platform can be used to build training datasets for RL agents, as well as a baseline solution to evaluate their performance.

II. RELATED WORKS

A. Trajectory Planning for Manipulators

We proposed TOCALib, an original approach based on creating a precomputed motion library for manipulators using the FROST package in MATLAB, solving nonlinear programming (NLP) tasks for various end-effector positions with IPOPT and considering a set of symbolic constraints (dynamics, kinematics, collisions). For collision avoidance,

¹Yulia Danik is with FRC CSC RAS, Moscow, Russia and MIPT, Dolgoprudny, Russia yuliadanik@gmail.com

²Dmitry Makarov is with FRC CSC RAS, Moscow, Russia and MIPT, Dolgoprudny, Russia makarov@isa.ru

³Aleksandra Arkhipova is with Robotics Center, Sberbank of Russia, Moscow, Russia alsearkhipova@sberbank.ru

⁴Sergei Davidenko is with the Robotics Center, Sberbank of Russia, PhD student in Skolkovo Institute of Science and Technology, Moscow, Russia Sergei.Davidenko@skoltech.ru

⁵Aleksandr Panov is with the AIRI and Moscow Institute of Physics and Technology, Moscow, Russia panov@airi.net

¹<https://sites.google.com/view/tocalib?usp=sharing>

the Differentiable Collisions library in Julia is separately integrated, allowing distance evaluation between primitives and obtaining the first derivative of this distance. TOCALib approach is positioned as a means of obtaining a high-quality dataset with manipulator trajectories. Table I presents a comparison of this approach with most popular alternative methods based on different criteria. Thus, the proposed approach provides high computational efficiency and physical accuracy through symbolic optimization and consideration of the full dynamic model. These properties make it especially suitable for complex industrial tasks and the generation of high-quality data for reinforcement learning (RL).

B. Precomputed motion libraries

There are no existing works in the literature dedicated to creating motion libraries for manipulators; however, this idea has shown promising results for bipedal robots. For instance, in [5–7], a library of gaits with various longitudinal and lateral speeds was created for the Cassie robot and full-body humanoid robot [8] using the C-FROST framework [9]. If a specific solution was absent in the library, it was obtained using trilinear interpolation. FROST and its extension, C-FROST, are frameworks that solve control problems for hybrid systems (i.e., systems with both continuous and discrete subsystems) using the hybrid zero dynamics method [10]. To accelerate optimization, gradients of kinodynamic and other constraints are computed symbolically.

C. Collision Detection

The most widely used collision detection algorithms based on Gilbert-Johnson-Keerthi (GJK)[11, 12] and Minkowski Portal Refinement (MPR)[13, 14] methods, which use support mapping to determine collisions between convex objects. Enhancements to MPR and zero-crossing handling functions are proposed in [15]. These methods are implemented in the Flexible Collision Library (FCL) [16] and physics engines such as Bullet [17], MuJoCo [18] etc. However, they are not differentiable due to logical control flow.

Differentiable Collision (DCOL) method is implemented in the Julia DifferentiableCollisions library [19]. It determines collisions between convex primitives, including polyhedra, capsules, cylinders, cones, ellipsoids, and padded convex polygons. These shapes can approximate the geometry of the robot and any surrounding objects. DifferentiableCollisions computes the scaling factor of convex bodies at which a collision occurs. If the scaling factor is greater than one, no collision is present. The library also computes the derivative of this factor with respect to the geometric parameters of convex bodies, such as position, orientation, and size, making it suitable for gradient-based optimization methods. Thus, it has greater computational efficiency than alternative methods.

III. PROBLEM STATEMENT

We consider manipulator movement with collision avoidance as a nonlinear programming problem (NLP). To describe the dynamics of the whole robot the next second-order

Euler-Lagrange equation is used

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Bu, \quad (1)$$

where $q \in \mathbb{R}^n$ are the generalized coordinates, $M(q)$ is the mass matrix, $C(q, \dot{q})\dot{q}$ are vectors containing the centrifugal and Coriolis forces, $G(q)$ are the gravitational forces, and $B \in \mathbb{R}^{n \times m}$ is the actuation matrix, $u \in U \subset \mathbb{R}^m$ is the control (torques) actuating the system, U is the set of admissible control inputs. Let's rewrite (1) as

$$\dot{x} = f(x) + g(x)u$$

where $f(x) = \begin{bmatrix} \dot{q} \\ -M(q)^{-1}(C(q, \dot{q})\dot{q} + G(q)) \end{bmatrix}$, $g(x) = \begin{bmatrix} 0 \\ M(q)^{-1}B \end{bmatrix} u$, $x = [q^T \dot{q}^T]^T \in \mathbb{R}^{2n}$. The cost function:

$$\min_{x_i, u_i} \sum_{i=1}^N Cost(x_i, u_i, i), \quad (2)$$

there are the following constraints:

- *C1*. The closed system dynamics equation, $\dot{x} = f(x) + g(x)u(x)$,
- *C2*. The physical feasibility condition and fixed-joints constraints (holonomic), including the torque limits, joints angles and velocities limits, etc.,
- *C3*. Collision constraints,
- *C4*. Target constraints (end-effector final point or path).

In our work we use the sum of squares of controls as a cost function, that is $Cost = u^T u$. To construct the motion library, it is necessary to solve the problems 2 for the given sets of initial and final states, corresponding to the initial and final positions of the robot manipulators.

IV. MOTION LIBRARY DESIGN

A. NLP Solver Framework

The FROST framework was selected for solving the NLP, as it allows for the rapid formulation and solution of control problems for robotic systems. Additionally, it enables solving control problems for hybrid systems (e.g., walking robots) in the future. FROST is implemented in MATLAB and uses the direct collocation method for numerically solving the nonlinear dynamics *C1*. It supports multiple solvers. In the current work, IPOPT was used [20]. To run IPOPT from FROST, all NLP constraints must be translated into C language files and compiled using the mex compiler embedded in MATLAB. The resulting solution is not globally optimal and includes elements of stochastic search.

B. Constraints

The constraints related to the robot's dynamics and kinematics (i.e., constraints *C1* and *C2*) are automatically generated in FROST from the URDF file in symbolic form. The constraints *C3* and *C4* are user-defined based on the specific requirements of the task. FROST does not have a convenient collision detection and avoidance mechanism that is why additional third-party packages need to be used.

TABLE I
COMPARISON OF APPROACHES TO CREATING A PRECOMPUTED MOTION LIBRARY FOR MANIPULATORS

Criterion	TOTALib	MoveIt / Tesseract	RoboDK	RMP Flow ^a /TrajOpt ^b /CHOMP ^c	OMPL
References	This paper	https://moveit.ai/ https://github.com/tesseract-robotics/tesseract-ocr/tesseract	https://robodk.com/	See below the table	https://ompl.kavrakilab.org
Approach Characteristics	Symbolic optimization; IPOPT solver for precise trajectories with dynamics and collision considerations; integration with Julia for distance gradient computation	Kinematic planning (OMPL: RRT, PRM); collision avoidance via TrajOpt and DART; tight integration with ROS	Offline programming for industrial tasks	Numerical optimization (SQP, Adam); FCL and Bullet for reactive collision avoidance; efficient in real-time	Heuristic path construction (PRM, RRT), user-defined constraint support
Planning Methods	Nonlinear programming (NLP) with IPOPT and symbolic derivatives	Sampling-based planning with collision constraints (RRT, PRM)	Calculation of static trajectories without optimization	Numerical trajectory optimization (SQP, Adam)	Sampling-based planning (RRT, PRM)
Collision Algorithm	Differentiable Collisions (Julia), symbolic constraints	FCL (Flexible Collision Library)	Simple collision checking for offline tasks	Bullet / FCL (reactive collision avoidance)	FCL for basic collision checking
Dynamic Consideration	Full (forces, torques, and acceleration constraints)	None (kinematics only)	None (only basic motion constraints)	Yes (acceleration and velocity consideration)	None, focused on static planning
Constraint Handling	Symbolic constraints for collisions and dynamics	Collisions and motion boundaries	Range of motion constraints	Collisions, speeds, and accelerations	Basic collision prevention constraints
Trajectory Accuracy	High, symbolic derivatives and dynamic constraints	Medium, limited by kinematic parameters	High for industrial tasks with fixed trajectories	High, adaptive trajectories for dynamic conditions	Low, primitive heuristics
Adaptability to Environmental Changes	Limited, designed for precomputed trajectories	Limited, suitable for static tasks	Low, designed for predefined motions	High, adaptive real-time response	Low, focused on tasks without dynamics
Symbolic Constraint Support	Yes, supported in Julia, useful for gradient optimization	No, numerical methods only	No, designed for static tasks	No, numerical methods and adaptive correction	No, heuristic constraints only
Use in RL	Suitable for generating high-quality RL training data	Moderate, used as initial dataset	Low, designed for fixed tasks	High, suitable for adaptive RL	Low, limited RL support
Computational Resource Intensity	High, due to IPOPT and symbolic optimization, but justified for complex tasks	Moderate, optimized for ROS, suitable for research	Low, simple maintenance for static tasks	High, especially in real-time tasks	Low, focused on simple calculations

^ahttps://docs.omniverse.nvidia.com/isaacsim/latest/concepts/motion_generation/rmpflow.html

^b<https://github.com/tesseract-robotics/trajopt?ysclid=m3hqz7burk86695758>

^c<https://personalrobotics.cs.washington.edu/publications/zucker2013chomp.pdf>

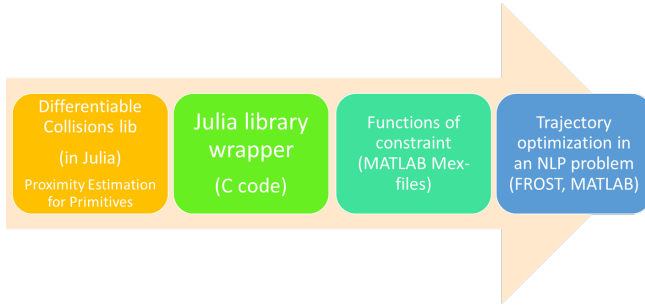


Fig. 1. Adding a Collision Avoidance Constraint to the Aloha Manipulator Trajectory Optimization Problem

C. Collision Detection Tool

To handle collisions, the DifferentiableCollisions (DCOL) library implemented in Julia was used. It also provides the first derivative of the collision parameters with respect to object position and orientation, making it suitable for integration into gradient-based control and learning methods.

For convex primitive shapes, collision constraints are generated in symbolic form using the DCOL, written in Julia. For DCOL, a wrapper is generated in C, which is compiled into a MEX file, accessible for calling from FROST (MATLAB, IPOPT). The process of adding constraints to the

NLP is shown in Fig. 1.

D. Interpolation

Trilinear interpolation is employed as an alternative to solving complex optimal control problems, using 5th-order Bézier polynomials to describe the trajectories in the 8 nearest points. The coefficients of the polynomials for the stored trajectories are involved in the interpolation and help to determine an approximate solution to the problems not included in the library. Trilinear interpolation is a method of multivariate interpolation on a 3-dimensional regular grid. It approximates the value of an intermediate point (x, y, z) within the local axial rectangular prism linearly, using data on the lattice points. An example of interpolation errors is shown in Fig. 2.

V. EXPERIMENTS

To obtain an accurate mathematical model we have firstly identified the robot parameters (Table II).

Secondly, we have experimentally verified that the error between the real trajectories obtained using the build-in Aloha controller and the trajectories simulated using mathematical models is quite small (Fig.3). The average error

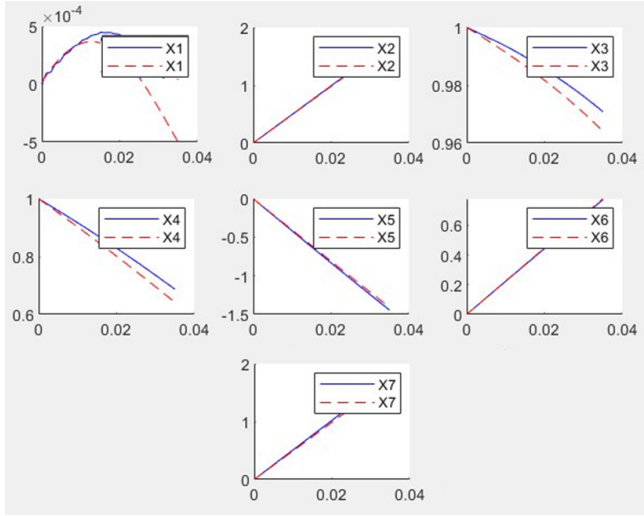


Fig. 2. Interpolation errors for all joints of the Aloha manipulator when moving the manipulator from a given initial point to a final point located 40 cm away. The red dashed curve represents the interpolation results, while the solid blue line shows the exact solution obtained through optimization.

TABLE II
JOINT EFFORTS, VELOCITIES, AND BOUNDS (DEGREES)

Name	Effort (Nm)	Velocity (rad/s)	Bounds (rad)
joint1	9	3.7699	[-2.11...3.1294]
joint2	9	3.7699	[0.044...3.65]
joint3	9	3.7699	[0.0364...3.229]
joint4	3	12.5664	[-1.37...1.36]
joint5	3	12.5664	[-1.534...1.541]
joint6	3	12.5664	[-2.1...2.087]
joint7	3	12.5664	[0.02...14.35]
joint8	3	12.5664	[0.02...14.35]

of the joints angles is 0.0258 rad, the average error of the end-effector position in meters is 0.0170.

For the numerical experiments with TOCALib, the NLP problems (2) were solved, considering the full model of Mobile Aloha with 39 DOF, i.e., $q \in \mathbb{R}^{39}$. The cost function was defined as the sum of the squared controls, and the entire time interval was divided into 31 segments (i.e. $N = 31$).

A. Enhanced URDF Loading Mechanism

The original URDF model of ALOHA was initially written as a single, monolithic file. The authors introduced a parameterization of the URDF model fields by using XACRO files and implemented a modular system². As a result, the model became more adaptable to changes in manipulator configurations, and code duplication was eliminated, significantly enhancing its flexibility and maintainability.

B. Self-collision Constraints

Each Aloha robot manipulator consists of 8 links, each represented by a separate capsule (highlighted in transparent blue), resulting in a total of 16 capsules for the two manipulators.

²<https://sites.google.com/view/tocalib?usp=sharing>

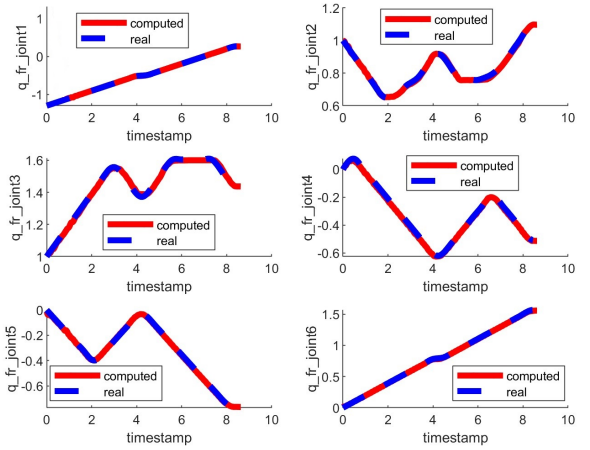


Fig. 3. The comparison of real and computed joint angle trajectories

The total number of 141 self-collision constraints were used, including the self-collisions of each manipulator, the collision of one manipulator with the other and, finally, the collision of manipulators with the base. A flexible mechanism was implemented for adding collision constraints between specific capsules. In the optimization process, a holonomic constraint $\alpha(q)_i > 1$, $i = 1, 2, \dots, 141$ was applied, where α is the scaling parameter for the capsules. For gradient optimization, the Jacobian for each $\alpha(q)_i$ is computed as follows

$$\left[\frac{\partial \alpha_i}{\partial q} \right]_{1 \times 39} = \left[\frac{\partial \alpha_i}{\partial \mathbf{params}} \right]_{1 \times 14} \left[\frac{\partial \mathbf{params}}{\partial q} \right]_{14 \times 39}, \quad (3)$$

where \mathbf{params} is the parameter vector for two examined capsules which contains 3D position vectors and 4D quaternions defining orientation (a vector of length 7 for each capsule).

C. Collision with Static Objects

For experiments, we considered the approximation of collision objects using spheres and polytopes. Each sphere is described by a radius, a position and an orientation vector. A polytope is described by a set of halfspace constraints $Aw \leq b$, where $w \in \mathbb{R}^3$, $A \in \mathbb{R}^{m \times 3}$ and $b \in \mathbb{R}^m$ represent the m halfspace comprising the polytope (in our experiments we used four parallelepipeds described by $m = 6$ halfspaces). A polytope also has position and orientation parameters. Thus, one additional constraint was added for collision of robot links with a sphere object $\alpha(q)_i > 1$, $i = 1, 2, \dots, 16$ (16 pairs, 8 for each manipulator) and another for collision with a shelf constructed from 4 polytopes (64 pairs for collision check).

D. Tasks

The numerical experiments involved moving manipulators along a number of points with self-collision avoidance and collision-free moving in the presence of static objects (a sphere (Fig. 4-6) and a shelf (Fig. 7,9,10,13)) and a moving sphere (Fig. 11). The optimization tasks incorporated motion

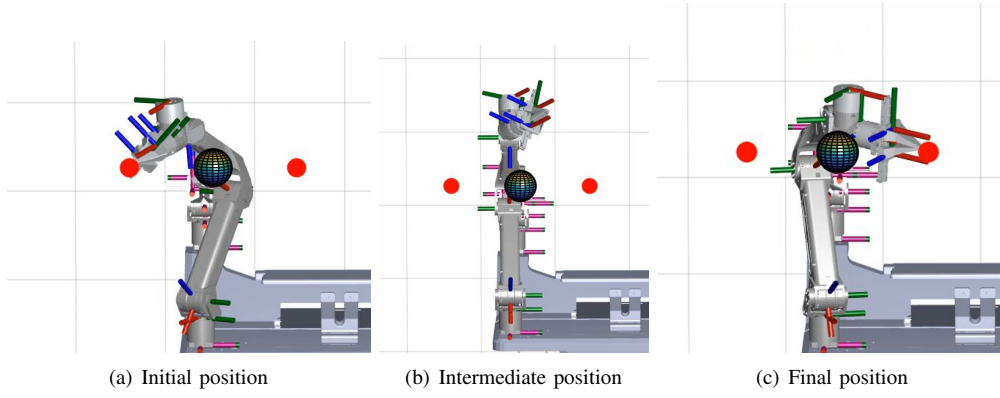


Fig. 4. Collision-free trajectories of a manipulator

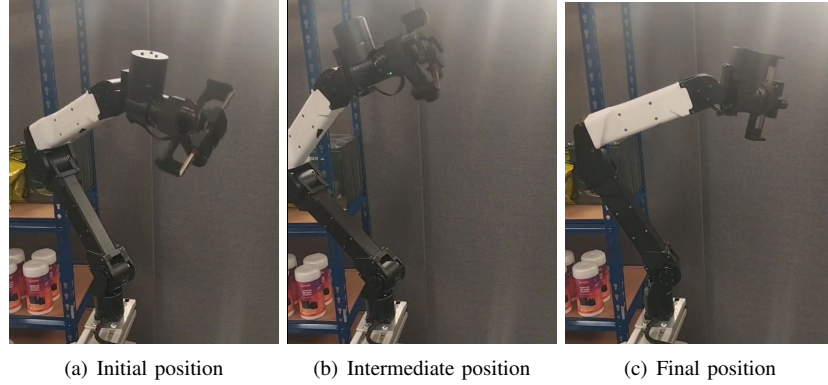


Fig. 5. The experiment on a real Aloha (with a virtual sphere)

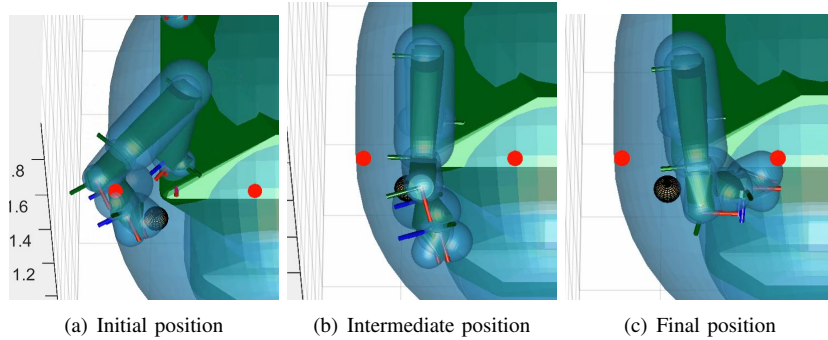


Fig. 6. The collision of the manipulator with the sphere without collision constraints

equations, self-collision constraints, and target end-effector positions. Fig. 4 corresponds to the scenario with a sphere avoidance. The goal was to follow the way-points (red dots) without collisions. The obtained trajectory was also tested on a real Aloha as shown in Fig. 5. If the simulation is carried out with the turned-off collision constraints the manipulator collides with the sphere (Fig. 6).

We have also conducted the experiments with a moving sphere (Fig. 11). Robot successfully avoids it when optimization is made with our collision constraints and collides with it otherwise (Fig. 12).

The experiment with a more complicated object (Fig. 7) showed that the proposed algorithm found a collision-free trajectory and the corresponding control. We compared

TOCALib with CHOMP. In contrast, CHOMP algorithm provides only the way-points. To calculate the trajectories it approximates obstacles with spheres (Fig. 7) and finds a smooth collision-free trajectory. Though, the direct comparison of TOCALib and CHOMP is not possible, as CHOMP does not account for dynamics of the robot, the qualitative comparison of the obtained trajectories was made.

Two manipulator libraries were constructed using TOCALib. The first one contains 36 goal positions within a shelf ($x \in [0.5, 0.55, 0.6]$, $y \in [0.55, 0.1, 0.15]$, $z \in [0.95, 1.05, 1.15]$). Our method succeeded (Fig. 9) in 90% of cases (in 10% there was no solution because of the collision in the final state), while CHOMP managed to find the solution for 44% of cases, not including cases where it

failed to give any result and the cases where the obtained solution was infeasible (false positive) because the integrity of the robot was compromised (Fig. 8). Moreover, we had to turn off the self-collision option in CHOMP, otherwise it provided no solution.

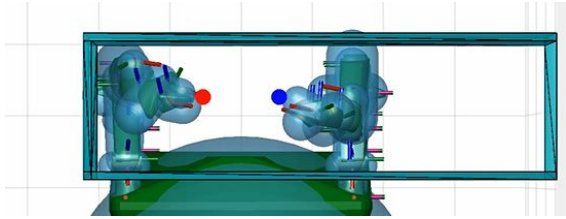


Fig. 7. Collision avoidance for a shelf object

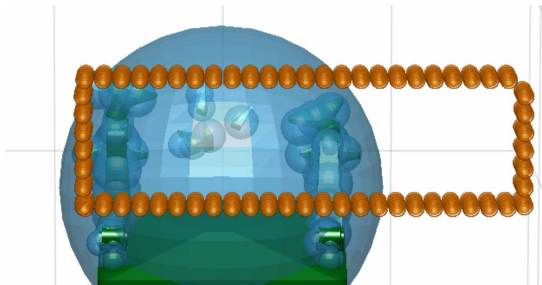


Fig. 8. Approximation of a shelf in CHOMP (failed to provide a feasible trajectory: the connection of the arms is broken)

The second library was constructed for an environment with a shelf placed between manipulators and 27 target positions (Fig. 13): some of them must be reached only by one manipulator, and others must be reached by both manipulators consistently. Thus, the library includes 325 combinations of final positions for both manipulators (cases where both manipulators have the same goal positions were not considered). From 325 trajectories calculated using TOCALib (Fig. 10) and CHOMP 159 were infeasible (a task for which no solution was obtained using TOCALib or CHOMP was considered unacceptable). The table III shows the result only for feasible cases. The CHOMP operation time was limited to 600 seconds, which corresponds to the maximum operating time limit of our method (see the section Simulation and runtime parameters). The algorithm for solving the problem using CHOMP is given in the APPENDIX.

As can be seen from the table, our approach solves the problem in a significantly larger number of cases, while spending approximately the same time as CHOMP. Moreover, in 22 % cases the video of CHOMP trajectories shows unexpected behavior (Fig. 14), thus its real success rate is likely even lower (we did not exclude such solutions from the list of successful ones for CHOMP).

E. Simulation and runtime parameters

During the simulation the end-effector target position error is set to 10^{-3} and the permissible deviation from the dynamic equation in (1) is 10^{-4} . On average, trajectory computation

TABLE III
RESULTS OF THE EXPERIMENTS WITH LIBRARY

Method	Success	Success rate	Average time, sec
TOCALib	129 of 166	78%	483,55
CHOMP	85 of 166	51%	465,21

takes approximately 2 minutes. The maximum computation time for one trajectory is set to 10 minutes. Computations were carried out on Intel Core i7 processor with 16 GB RAM.

VI. CONCLUSIONS

The proposed Two-Arm Optimal Control and Avoidance Library (TOCALib) framework successfully combines pre-computed motion strategies with collision avoidance techniques to address the challenges of bimanual manipulation. The integration of optimisation in FROST with DifferentiableCollisions method implemented in Julia for collision detection ensures the generation of high-quality motion trajectories in complex environments that are both accurate and safe. The collision constraints are symbolically represented in the optimization problem together with kinodynamic constraints and enable the integration of efficient collision avoidance into the trajectory planning process. TOCALib offers several advantages, such as support for a wide range of robots, fast optimization using gradient-based methods, and flexible collision-checking control. Approximate solutions can also be obtained without full optimization by using interpolation which gives adaptability. TOCALib works both for static and dynamic environments.

In our comprehensive evaluation, we demonstrated the significant advantages of TOCALib over CHOMP for trajectory planning in bimanual manipulation tasks. While CHOMP provides only waypoints without accounting for robot dynamics and approximates obstacles with spheres, TOCALib offers a complete solution with full consideration of kinodynamic constraints. Our experimental comparison using a library containing 166 feasible goal positions showed that TOCALib successfully solved 129 cases (78% success rate), while CHOMP managed only 85 cases (51% success rate).

However, TOCALib has limitations. The computational time and limited number of allowed approximation primitives restrict TOCALib applicability in real-time scenarios that require rapid and accurate adaptation to environmental changes. Nevertheless, our approach provides the interpolation tool, which helps to overcome this problem by utilizing precomputed optimal feasible trajectories with close goal positions. Moreover, our method provides a powerful solution for creating high-quality datasets for reinforcement learning and lays the foundation for generating diverse scenarios necessary for training RL agents. The experiments demonstrated the effectiveness of TOCALib for bimanual manipulators.

The directions of future research include the application of the method for other types of robots, implementation of parallel calculations and exploration of alternative interpolation methods.

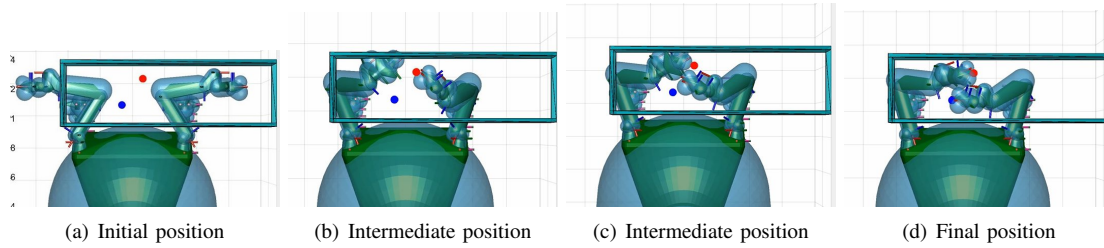


Fig. 9. Self collision avoidance in a presence of a shelf

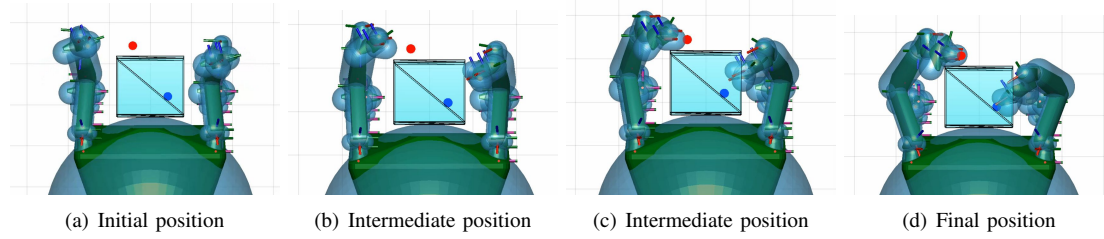


Fig. 10. One trajectory from a library with shelf avoidance

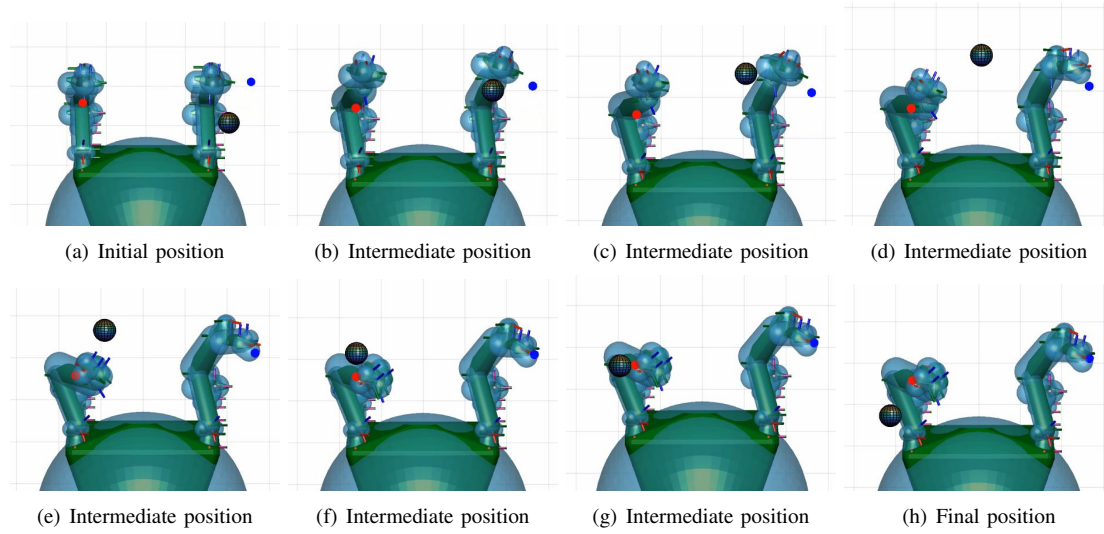


Fig. 11. Collision avoidance of a moving sphere

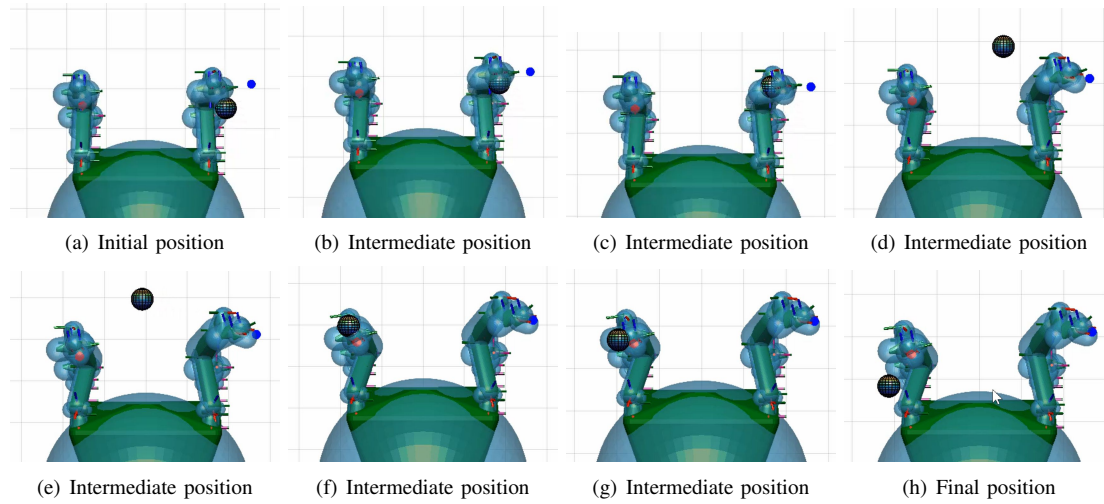


Fig. 12. Collision with a moving sphere without collision constraints

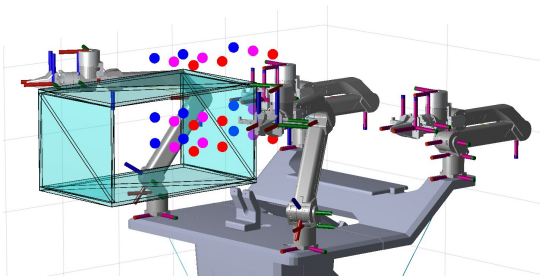


Fig. 13. The distribution of goal points in a library (red for the left arm, blue - for the right arms, and magenda - for both arms)

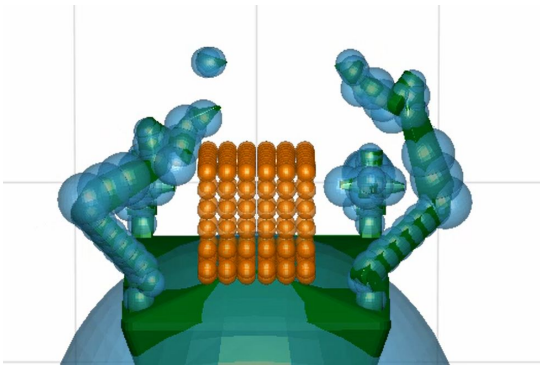


Fig. 14. The trajectory obtained by CHOMP with shelf collision avoidance (one connection of the arm is broken)

REFERENCES

- [1] Jordi Pages, Luca Marchionni, and Francesco Ferro. "Tiago: the modular robot that adapts to different research needs". In: *International Workshop on Robot Modularity, IROS*. Vol. 290. 2016.
- [2] Tom Sorell. "Cobots, "collaboration" and the replacement of human skill". In: *Ethics and Information Technology* 24.44 (2022).
- [3] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. *Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation*. 2024. arXiv: 2401.02117 [cs.RO].
- [4] Jianlan Luo et al. *SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning*. 2024. arXiv: 2401.16013.
- [5] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. *Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation*. 2024. arXiv: 2401.02117 [cs.RO].
- [6] Jenna Reher and Aaron D. Ames. *Inverse Dynamics Control of Compliant Hybrid Zero Dynamic Walking*. 2020. arXiv: 2010.09047.
- [7] Jenna Reher and Aaron D. Ames. *Control Lyapunov Functions for Compliant Hybrid Zero Dynamic Walking*. 2021. arXiv: 2107.04241.
- [8] E. Chaikovskaya et al. "Benchmarking the Full-Order Model Optimization Based Imitation in the Humanoid Robot Reinforcement Learning Walk". In: *2023 21st International Conference on Advanced Robotics (ICAR)*. IEEE. 2023, pp. 206–211. DOI: 10.1109/ICAR58858.2023.10406818.
- [9] Ayonga Hereid et al. *Rapid trajectory optimization using C-FROST with illustration on a Cassie-series dynamic walking biped*. 2021. arXiv: 2107.04241.
- [10] J.W. Grizzle, G. Abba, and F. Plestan. "Asymptotically stable walking for biped robots: analysis via systems with impulse effects". In: *IEEE Transactions on Automatic Control* 46 (2001), pp. 51–64.
- [11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal of Robotics and Automation* 4.2 (1988), pp. 193–203.
- [12] S. Cameron. "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra". In: *Proceedings of International Conference on Robotics and Automation*. 1997, pp. 3112–3117.

- [13] Michael J. and Newth O. *Minkowski Portal Refinement and Speculative Contacts in Box2D*. 2013.
- [14] X. Wang, J. Zhang, and W. Zhang. "The distance between convex sets with Minkowski sum structure: application to collision detection". In: *Computational Optimization and Applications* 77 (2020), pp. 465–490.
- [15] A. Neumayr and M. Otter. "Collision handling with variable-step integrators". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. 2017, pp. 9–18.
- [16] J. Pan, S. Chitta, and D. Manocha. "FCL: A general purpose library for collision and proximity queries". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3859–3866.
- [17] E. Coumans. *Bullet Physics Simulation*. 2015.
- [18] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.
- [19] K. Tracy. *DifferentiableCollisions.jl*. [Online]. Available: <https://github.com/kevin-tracy/DifferentiableCollisions.jl>. 2024.
- [20] A. Wächter and L. T. Biegler. "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming* 106.1 (2006). (preprint), pp. 25–57.

VII. APPENDIX

In this appendix, we present the algorithm 1 used for trajectory planning with CHOMP. The *CHOMPTrajectoryPlanning* procedure contains 3 main stages:

- finding a feasible trajectory for the left manipulator (steps 1-3),
- finding a feasible trajectory for both manipulators (steps 4-7),
- checking for self-collision between the two manipulators (step 8).

This decomposition allows for filtering out inherently infeasible trajectories. The procedure also includes a time limit for finding a single trajectory, as this can be a time-consuming process. A time limit of 600 seconds was chosen.

CHOMPTrajectoryPlanning utilizes a grid search over Euler angles to identify feasible orientations for end-effectors using *GetFeasibleState* procedure (algorithm 2). *GetFeasibleState* returns a feasible final position that does not lead to collision with the shelf, using inverse kinematics.

Algorithm 1 Trajectory Planning for Dual Manipulators using CHOMP

```
1: procedure CHOMPTRAJECTORYPLANNING( $x_0, p_{left}, p_{right}$ )
2:   Input: Initial robot position  $x_0$ , final positions of manipulators,  $p_{left}, p_{right}$ 
3:   Output: Waypoints (trajectory for  $x$ ), status, execution time, video
4:   start_time  $\leftarrow$  CURRENT_TIME()
5:   max_time  $\leftarrow$  600 ▷ maximum execution time in seconds
6:   status  $\leftarrow$  0
7:    $\Delta \leftarrow 0.4 \dots$  ▷ step size for Euler angles grid
8:   while (CURRENT_TIME() - start_time < max_time) and (status = 0) do
9:     Step 1: Generate orientation (Euler angles) for the left manipulator
10:    for  $e_{x_1}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
11:      for  $e_{y_1}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
12:        for  $e_{z_1}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
13:           $a_1 \leftarrow [e_{x_1}, e_{y_1}, e_{z_1}]$ 
14:          Step 2: Inverse kinematics and collision check for left manipulator
15:          [status_left,  $x_{f\_left}$ ]  $\leftarrow$  GETFEASIBLESTATE( $a_1, p_{left}$ , "left")
16:          if status_left = 0 then
17:            continue ▷ next iteration
18:          end if
19:          Step 3: Apply CHOMP algorithm for the left manipulator
20:          status_chomp_left  $\leftarrow$  CHOMP( $x_0, x_{f\_left}$ )
21:          if status_chomp_left = 0 then
22:            continue ▷ next iteration
23:          end if
24:          Step 4: Generate orientation for right manipulator
25:          for  $e_{x_2}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
26:            for  $e_{y_2}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
27:              for  $e_{z_2}$  from  $-\pi$  to  $\pi$  with step  $\Delta$  do
28:                 $a_2 \leftarrow [e_{x_2}, e_{y_2}, e_{z_2}]$ 
29:                Step 5: Inverse kinematics for right manipulator
30:                [status_right,  $x_{f\_right}$ ]  $\leftarrow$  GETFEASIBLESTATE( $a_2, p_{right}$ , "right")
31:                if status_right = 0 then
32:                  continue
33:                end if
34:                Step 6: Form combined final position vector for both arms
35:                 $x_f \leftarrow [x_{f\_left}, x_{f\_right}]$ 
36:                Step 7: Apply CHOMP algorithm for both manipulators
37:                status_chomp_both  $\leftarrow$  CHOMP( $x_0, x_f$ )
38:                if status_chomp_both = 0 then
39:                  continue
40:                end if
41:                Step 8: Check entire trajectory for self-collisions
42:                status_self_collision  $\leftarrow$  CHECKSELFCOLLISIONS(trajectory)
43:                if status_self_collision = 1 then
44:                  status  $\leftarrow$  1
45:                  waypoints  $\leftarrow$  trajectory
46:                  break ▷ exit all loops
47:                end if
48:              end for
49:            end for
50:          end for
51:        end for
52:      end for
53:    end for
54:  end while
55:  execution_time  $\leftarrow$  CURRENT_TIME() - start_time
56:  if status = 1 then
57:    video  $\leftarrow$  RECORDVIDEO(waypoints)
58:  end if
59:  return waypoints, status, execution_time, video
60: end procedure
```

Algorithm 2 Procedure for Inverse kinematics and collision check with a polytope

```
1: procedure GETFEASIBLESTATE( $a, p$ , manipulator)
2:   Input:  $a$  orientation ( $[e_x, e_y, e_z]$  euler angles),  $p$  position  $[x, y, z]$ , manipulator name (left or right)
3:   Output: status and  $x_f$ 
4:   Apply inverse kinematics
5:    $x_f \leftarrow \text{INVERSEKINEMATICS}(a, p, \text{manipulator})$ 
6:   if  $x_f$  does not exist then
7:     return false, NULL
8:   end if
9:   Check collision with polytope
10:  collision  $\leftarrow \text{CHECKPOLYTOPECOLLISION}(x_f)$  ▷ using Julia lib and capsules approximation
11:  if collision = TRUE then
12:    return false, NULL
13:  else
14:    return true,  $x_f$ 
15:  end if
16: end procedure
```
